IT3105 Project 2

Aksel Reiten
Magnus Moan

October 18, 2016

Exercise 2

a)

In this problem where the Q-function is fixed, it would be sufficient to use a single dictionary or list to represent the function. Still, since implementing assignment 3 through 5 demands changing the Q-function for the various states, we have chosen to represent the Q-function in a list of dictionaries instead. Thus, each position on the board has a dictionary with the Q-function associated to it. Our choice of using a dictionary over a list makes it easier for us to reason about our code, but it does not yield any other performance gains.

c)

If we use a greedy policy, our agent will choose action south initially. Over time we will see a decrease in the Q-value for south (in all states) since we most likely will end up in a hole in some future by choosing south. At some point we will have Q-values for the other directions that exceeds the Q-value of south in some state, and that other direction will be chosen instead. We will then start seeing a change in the Q-value for the other directions as well.

By using an ϵ-greedy strategy we will most likely (depending on our ϵ value) start exploring other actions much earlier than in the greedy strategy. This could initially be an advantage. However, when we are starting to learn a decent Q-function, we would want our selection policy to choose the greedy choice more often (since we've learned that this is a good choice). This dynamic of first exploring the space and then sticking to our learned strategy is often referred to as exploring and exploiting. This can be implemented by using a ϵ value that decreases over time, and therefore reduces exploring. Summarized: An ϵ-greedy policy presents an opportunity for learning a better Q-function, if we are careful in our ϵ choice. If it's too high we will end up exploring too much, and not learning anything useful, and if it's too low we will not explore enough. The ϵ-value should also decrease over time, meaning we prefer exploiting over exploring when the learning process nears its end.

d)

$Q(s, South)$ is the expected value (cumulative discounted rewards) of going south when in state s. $Q(s,a)$ for any state-action pair is a weighted sum of accumulated immediate rewards. In statistical terms we are looking at a stochastic Markov chain. Each episode will update $Q(s,a)$ for some state-action pairs, based on accumulated immediate rewards. In the long run $Q(s,a)$ will tend to its *real* value, or what is usually called a steady state. By updating Q-values using these accumulated immediate rewards we will move towards the steady-state. When our Q-function reaches a steady-state we will know the true expected value of choosing action a in state s, which again will make it easier to choose the right action in any given state. When using a straight up greedy action choosing policy we will most likely move more slowly towards the steady state compared to a ϵ-gready strategy (given a smart ϵ-greedy strategy as described in Exercise 2c).

Exercise 3

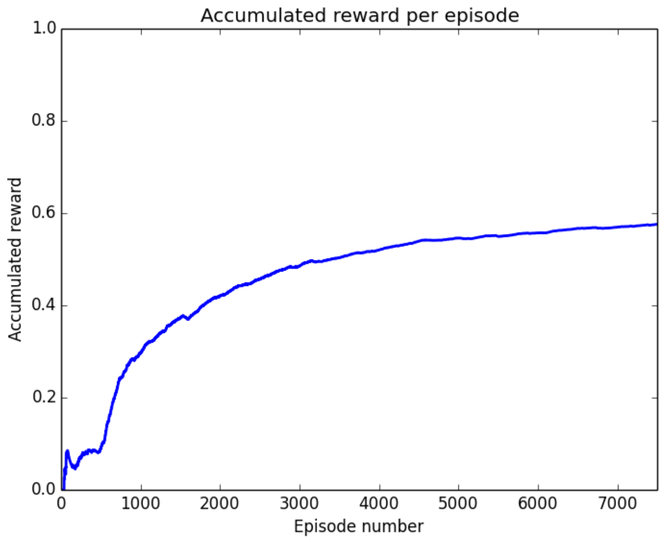

Figure 1: Accumulated reward per episode for the Frozen Lake problem

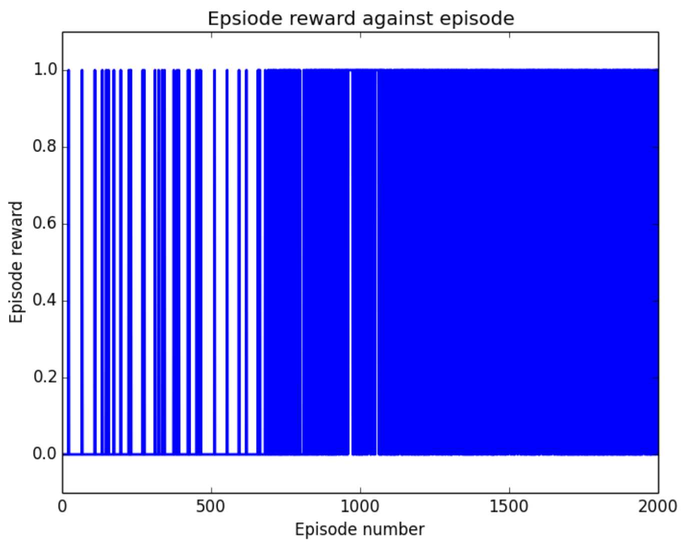

Figure 2: Reward per episode for the Frozen Lake problem

Exercise 4

Q-learning uses an off-policy learning technique. This means that it learns by estimating its expected future return based on a greedy action choosing policy. However, in practice we implement an ϵ-greedy strategy, which does not necessarily pick the greedy choice. In other words, our update policy is not the same as our action choosing policy, and we say that Q-learning is off-policy. $Q(s, a)$ is updated in the following way:

$$Q(s,a) = Q(s,a) + \mu(r + \lambda \max_{a'} Q(s', a') - Q(s,a)) \tag{1}$$

When we now instead update our Q-function based on our actual action choosing policy, we say that we are using an on-policy. More specifically we've implemented the SARSA (State-Action-Reward-State-Action) algorithm. In this case, $Q(s, a)$ is updated in the following way:

$$Q(s,a) = Q(s,a) + \mu(r + \lambda Q(s', a') - Q(s,a)) \tag{2}$$

Exercise 6

Yes, we can definitively see the human agent as a policy that generates data an artificial agent can learn from. The human agents will have some level of experience regarding how to interact with the different customers. By using this policy to train the artificial agent we will most likely not end up in a situation where many customers are dissatisfied. However, it should be noted that human agents are prune to error, and the policy they think is best might not always be the best policy. We should therefore be careful when using human-made policies as training data. Hence, it would be wise to allow for some sort of randomness in our training algorithm, to break out of possible wrong policies generated by the human agents. Based on this we would recommend using an on-policy learning technique, given that our action choosing policy contains some sort of randomness.

Exercise 7

In this project, we focused on writing code that could easily implement both Taxi and Frozen Lake. Thus, we could reuse the data structure from Exercise 2 - 6, and use the list of dictionaries in this problem as well.

The following graph illustrates the performance of our algorithm, by plotting the total reward per episode against the episode number:

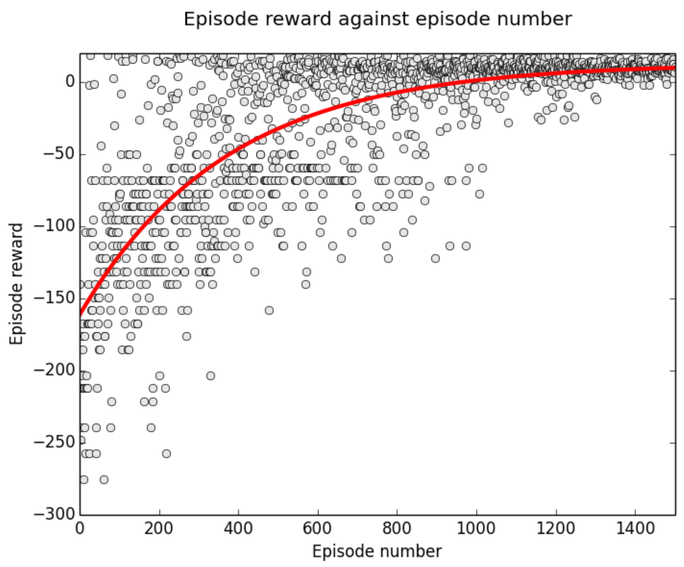

Figure 3: Total reward per episode against episode number for the Taxi-problem